Attorney Docket No. IBM/209
Confirmation No. 3405

PATENT

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

Ex parte William Joseph Armstrong, Chris Francois, Naresh Nayar

---

Appeal No. _____
Application No. 09/939,235

APPEAL BRIEF

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	William Joseph Armstrong et al.	Art Unit:	2195
Application No.:	09/939,235	Examiner:	Kenneth Tang
Filed:	August 24, 2001	Atty. Docket No.:	ROC920010252US1
For:	YIELD ON MULTITHREADED PROCESSORS		

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

I. REAL PARTY IN INTEREST

This application is assigned to International Business Machines Corporation, of Armonk, New York.

II. RELATED APPEALS AND INTERFERENCES

An appeal is currently pending in an application related to the instant application, U.S. Patent Application No. 09/939,232, also filed on August 24, 2001 by William Joseph Armstrong et al., entitled "SYSTEM FOR YIELDING TO A PROCESSOR".

III. STATUS OF CLAIMS

Claims 1-27 and 31-33 are pending in the Application. Claims 1-27 and 31-32 stand rejected, and are now on appeal. Claim 33 was added, but never addressed by the Examiner. Claims 1 and 13 have each been amended once, and claims 28-30 have been canceled.

IV. STATUS OF AMENDMENTS

No amendments have been filed subsequent to the Final Rejection.

V. SUMMARY OF CLAIMED SUBJECT MATTER

Applicants' invention is generally directed to performing processor yields with multithreaded central processing units (multithreaded CPU's). The invention is more particularly directed to accommodating conventional yield calls for multithreaded CPU by coordinating yielding threads within the hypervisor. A yield command is a convention whereby executing entities share access to single-threaded CPU's. For instance, an idle executing entity, e.g., a virtual processor with no work to do, may relinquish its CPU availability in response to receiving a yield command from another executing entity with work to do.

To appreciate this invention, it is important to emphasize the differences between the claimed multithreaded CPU and a conventional single-threaded, or non-multithreaded CPU. A multithreaded CPU allows concurrent presence of multiple active threads on the same (multithreaded) CPU. That is, a multithreaded CPU is a processor chip that can execute two or more threads in hardware. While a conventional, non-multithreaded CPU may operate in an environment with multiple threads, only one thread is active at time on the non-multithreaded CPU. Further, any coordination in thread sequential execution as between the single-threaded CPU's in accomplished in software, not hardware (as with the multithreaded processor). (Application, p. 1, l. 15 to p. 2, l. 8).

Multithreaded CPU's generally provide greater processing efficiencies, but prior to the present invention, have been unable to use conventional yield commands (as can single-threaded CPU's). One reason for this is because multithreaded CPU's have a programmatic limitation (which single-threaded CPU's do not have) that requires all threads to execute within a common virtual space, e.g., a hypervisor or partition. The claimed subject matter enables a thread executing on a multithreaded CPU to yield, and does so in a manner to ensure that all threads in the multithreaded CPU execute within the same virtual space. For instance, a single thread's yield may be deferred until another thread is in a ready-to-yield state, which ensures that the threads can all yield in a common virtual space. (Application, p. 4, ll. 1-11 and p. 11, ll. 13-19).

More specifically, embodiments consistent with the principles of the present invention include an apparatus, method, and program product configured to facilitate the sharing of physical resources on a multithreaded CPU by coordinating the yielding of multiple threads

executing on the multithreaded CPU. The yield of a first thread may be deferred while it waits for at least a second thread of the CPU to become ready to yield. For instance, the embodiment may spin the first thread as it waits for other threads of the CPU to make yield calls. In response to the second thread becoming ready to yield, the first thread may yield, itself. More particularly, the embodiment may place at least the first and second threads of the processor on an idle loop. A program of the embodiment may additionally save the states of the operating system(s) executing the threads. Alternatively, a program of the embodiment may abandon the yield of the first thread while spinning and after detecting an event, such as a time-out or external I/O interrupt, that is related to the reason that initially required the yield in the first place. (Application, p. 4, l. 13 to p. 5, l. 5).

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

A. Claims 1-14, 16-27 and 31-32 stand rejected under 35 U.S.C. § 102(e) as being anticipated by U.S. Patent No. 6,195,676 to Spix et al. (*Spix*).

B. Claim 15 stands rejected under 35 U.S.C. § 103(a) as being unpatentable over *Spix* in view of U.S. Patent No. 5,978,830 to Nakaya et al. (*Nakaya*).

VII. ARGUMENT

Applicants respectfully submit that the Examiner's rejections of claims 1-27 and 31-32 are not supported on the record, and should be reversed. In summary, the Examiner has failed to appreciate the difference between a multithreaded CPU and a single-threaded, conventional CPU operating in a multithreaded environment. None of the cited prior art discloses or suggests a multithreaded CPU, among other claimed features.

In addition, Applicants note that claim 33 added in the Amendment and Response filed on February 24, 2005 depends from claim 16, but has not been addressed by the Examiner in either the Office Action mailed May 23, 2005 or the Final Office Action mailed December 12, 2005. In their Amendment and Response filed on September 23, 2005, Applicants brought the omission of a rejection to claim 33 to the attention of the Examiner, and also requested

reconsideration and allowance of the claim as a dependent claim of claim 16 (*See* Amendment and Response filed 09/23/2005, respectively at page 7, footnote 1, and page 10, line 25, both in the Remarks).

Applicants will hereinafter address the Examiner's rejections in the order presented in the Final Office Action. Within the discussion of each rejection, the various claims that are the subject of the Examiner's rejections will further be addressed in order, starting with the independent claims, and then addressing various dependent claims reciting additional subject matter that is distinguishable from the prior art of record. In some cases, specific discussions of particular claims are not made in the interests of streamlining the appeal. The omission of a discussion with respect to any particular claim, however, should not be interpreted as an acquiescence as to the merits of the Examiner's rejection of the claim, particularly with respect to claims reciting features that are addressed in connection with the rejections applied to other claims pending in the appeal.

A. Claims 1-14, 16-27 and 31-32 are novel over *Spix*.

The Examiner argues that *Spix* anticipates claims 1-14, 16-27 and 31-32. However, at least because *Spix* fails to disclose a multithreaded CPU, the rejections should be reversed. Anticipation of a claim under 35 U.S.C. §102 requires that "each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference." Verdegaal Bros., Inc. v. Union Oil Co., 2 USPQ2d 1051, 1053 (Fed. Cir. 1987), *quoted in* In re Robertson, 49 USPQ2d 1949, 1950 (Fed. Cir. 1999). Absent express description, anticipation under inherency requires extrinsic evidence that makes it clear that "the missing descriptive matter is necessarily present in the thing described in the reference, and that it would be so recognized by persons of ordinary skill." Continental Can Co. v. Monsanto Co., 20 USPQ2d 1746, 1749 (Fed. Cir. 1991), *quoted in* In re Robertson at 1951. "Inherency, however, may not be established by probabilities or possibilities. The mere fact that a certain thing may result from a given set of circumstances is not sufficient." Continental Can at 1749, *quoted in* In re Robertson at 1951.

As will be discussed in greater detail below, *Spix* does not disclose a multithreaded CPU and/or associated features as recited in claims 1-14, 16-27 and 31-32. As such, the rejections should be reversed.

Independent Claims 1, 16 and 31

The Examiner argues that *Spix* anticipates claims 1, 16 and 31. Applicants respectfully submit, however, that *Spix* fails to disclose each and every aspect recited in these claims, e.g., a multithreaded CPU, and therefore the Examiner's rejection under 35 U.S.C. §102(e) should be reversed. Applicants will discuss method claim 1 prior to discussing claims 16 and 31 (directed respectively to an apparatus and a program product).

In particular, claim 1 generally recites a method for sharing resources on a multithreaded CPU capable of executing a plurality of threads. The method includes deferring a yield of a first thread executing on the multithreaded CPU while waiting for at least a second thread executing on the multithreaded CPU to become ready to yield, and yielding the first thread in response to at least the second thread becoming ready to yield.

As such, this claim generally recites a method for sharing resources on a multithreaded CPU. As described on page 1-2 of the application, a multithreaded CPU is a processor chip that can execute two or more threads in hardware. The claimed invention specifically addresses programming constraints that have conventionally hindered multithreaded CPU resource sharing. For instance, all threads executing on a multithreaded CPU are required to execute within a common virtual space, such as a partition. The claimed invention enables resource sharing despite such constraints by deferring a yield of a first thread executing on the multithreaded CPU, while waiting for at least a second thread executing on the multithreaded CPU to become ready to yield, and yielding the first thread in response to at least the second thread becoming ready to yield.

Spix fails to teach this claimed feature. In fact, *Spix* is concerned only with single-threaded CPU's having a shared memory (col. 20, l. 28; col. 26, ll. 1-13; col. 15, ll. 35-41; and col. 16, ll. 21-30). Notwithstanding *Spix's* explicit descriptions of parallel, single-threaded

CPU's[1], there is not a single mention within *Spix* of a multithreaded CPU or its equivalent. There is consequently no requirement for threads in *Spix* to execute within a common virtual space, among other programming requirements of the claimed multithreaded CPU, and there it follows that there is no suggestion or teaching of a multithreaded CPU anywhere in cited reference.

Moreover, there is no teaching of a yield command in *Spix*. A yield has a definite meaning in the art, but a brief discussion of a yield is included to resolve any lack of clarity as perceived by the Examiner, and to highlight the distinctions between the present claims and the prior art.

In a conventional, non-multithreaded CPU partitioned environments, partitions share computing resources, such as CPU's. As explained in greater detail on pages 2-4 of the application as filed, CPU's within a conventional, non-multithreaded CPU environment are dynamically assigned to handle independent units of execution, in different partitions in the environment. These units of execution are often referred to as virtual processors, and in such environments, yield commands may be made by an operating system when a virtual processor of a partition cannot use its allocated CPU efficiently. For instance, a virtual processor may be in an idle loop or may have no work to do. The yield command causes the virtual processor to surrender its CPU availability to another virtual processor, and subsequently enter an idle state. Thus, the (non-multithreaded) CPU is used more efficiently by virtue of its allocation to a virtual processor that can utilize it.

While yield commands conventionally work well within multithreaded environments, e.g., where partitions share multiple non-multithreaded CPU's, yields have never before the present invention been used with multithreaded CPU's. One reason for this is because multithreaded CPU's have a programmatic limitation that requires all threads to execute within a common virtual space, e.g., a hypervisor or partition. The claimed subject matter enables a thread executing on a multithreaded CPU to yield, and does so in a manner to ensure that all threads in the multithreaded CPU execute within the same virtual space. Specifically, a single

---

[1]Though operating within a multithreaded environment, a single-threaded CPU is still not a multithreaded CPU.

thread's yield is deferred until another thread is in a ready-to-yield state, which ensures that the threads can all yield in a common virtual space.

While *Spix* does disclose one manner of managing resources, it still fails to disclose or suggest a yield command. Significantly, *Spix* does not once even mention the concept of a "yield" or its equivalent. The Examiner has failed to cite any portion of *Spix* where a yield is disclosed.

The (non-yield) methods disclosed in *Spix* are described at column 7, lines 36-45. The system of *Spix* attempts to schedule resources "by allowing each processor to access a single image of the operating system" to provide a "visual representation" for a plurality of compiling tools. To this end, the *Spix* system utilizes microprocessors (mprocs) in addition to a User-Side Scheduler (USS). In the text cited by the Examiner in column 8, the USS is described as prioritizing processes by allowing a mproc to grab a single image of the memory. The USS functions to place work and look for work in a request queue (column 8, lines 55-60). The USS further utilizes a scheme for marking data and/or resources (mark or gmark) (column 45, lines 37-48). In any case, the USS does not implement or suggest a yield command.

Because *Spix* discloses neither a multithreaded CPU nor a yield command, *Spix* fails to teach each and every element of claim 1. Applicants respectfully request that the §102(e) rejection of claim 1 be reversed.

Moreover, Applicants respectfully submit that claim 1 is also non-obvious over *Spix* as there is no suggestion in the reference, or elsewhere in the cited prior art, of performing yields using a multithreaded CPU. A *prima facie* showing of obviousness requires that the Examiner establish that the differences between a claimed invention and the prior art "are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art." 35 U.S.C. §103(a). Such a showing requires that all claimed features be disclosed or suggested by the prior art, along with objective evidence of the suggestion, teaching or motivation to combine or modify prior art references, as "[c]ombining prior art references without evidence of such a suggestion, teaching or motivation simply takes the inventor's disclosure as a blueprint for piecing together the prior art to defeat patentability -- the essence of hindsight." In re Dembiczak, 50 USPQ2d 1614, 1617 (Fed. Cir. 1999).

In the case of claim 1, the Examiner has provided no evidence of a motivation in the art to modify *Spix* to support yield commands for multithreaded CPU's. The absence of such a motivation and suggestion is partially attributable to the peculiarities associated with multithreaded CPU's, which are not accommodated in the *Spix* system. *Spix* has no appreciation for the need to check the state of one thread in connection with allocating another thread to handle a particular task. As such, *Spix* would not motivate one of ordinary skill in the art to defer a yield of one thread on a multithreaded CPU based upon the state of another thread. Accordingly, claim 1 is also non-obvious over *Spix*. Reversal of the Examiner's rejection of claim 1 is therefore respectfully requested.

Next, with respect to independent claims 16 and 31, which respectively recite an apparatus and a program product, each of these claims likewise recite yielding in connection with multithreaded processors. Accordingly, Applicants respectfully submit that claims 16 and 31 are novel and non-obvious over *Spix* and the other cited prior art of record for the same reasons as claim 1. Reversal of the Examiner's rejections, and allowance of claims 16 and 31, are therefore respectfully requested.

Dependent Claims 2 and 17

Dependent claims 2 and 17 respectively depend from claims 1 and 16, and each recites monitoring the plurality of threads being executed by the multiprocessor CPU for an occurrence. In rejecting these claims, the Examiner asserts that *Spix* discloses monitoring at col. 11, ll. 28-29. The cited text, however, concerns monitoring only in connection with a single-threaded CPU. Since *Spix* at least fails to disclose monitoring in connection with the claimed multithreaded CPU, Applicants respectfully submit that the Examiner has failed to establish anticipation of claims 2 and 17 by *Spix*. Reversal of the Examiner's rejections, and allowance of the claims, are therefore respectfully requested.

Dependent Claims 3 and 18

Dependent claims 3 and 18 respectively depend from claims 2 and 17, and each recites monitoring the plurality of threads being executed by the multiprocessor CPU for a spin lock or

idle loop. In rejecting these claims, the Examiner asserts that *Spix* discloses a general spin lock at col. 4, ll. 14-21. As above, however, the cited text concerns monitoring only in connection with a single-threaded CPU. Since *Spix* at least fails to disclose monitoring in connection with the claimed multithreaded CPU, Applicants respectfully submit that the Examiner has failed to establish anticipation of claims 3 and 18 by *Spix*. Reversal of the Examiner's rejections, and allowance of the claims, are therefore respectfully requested.

Dependent Claims 4 and 19

Dependent claims 4 and 19 respectively depend from claims 2 and 17, and each includes initiating a yield call in response to the occurrence. As above, the Examiner relies on *Spix* for the rejection, and as above, the cited text at best concerns yield processes in connection with a single-threaded CPU. Since *Spix* at least fails to disclose initiating a yield call in the context of the claimed multithreaded CPU, Applicants respectfully submit that the Examiner has failed to establish anticipation of claims 4 and 19 by *Spix*. Reversal of the Examiner's rejections, and allowance of the claims, are therefore respectfully requested.

Dependent Claims 5 and 20

Dependent claims 5 and 20 respectively depend from claims 1 and 16, and each recites marking storage of the first thread in response to receiving the yield call to indicate that the first thread is ready to yield on the multithreaded CPU. In rejecting these claims, the Examiner asserts that *Spix* discloses these processes at col. 45, ll. 37-53. The cited text, however, concerns a "Data Mark" process in connection with a single-threaded CPU. Since *Spix* at least fails to disclose marking storage in the context of the claimed multithreaded CPU, Applicants respectfully submit that the Examiner has failed to establish anticipation of claims 5 and 20 by *Spix*. Reversal of the Examiner's rejections, and allowance of the claims, are therefore respectfully requested.

Dependent Claims 6 and 21

Dependent claims 6 and 21 respectively depend from claims 1 and 16, and each recites spinning the first thread while waiting for at least the second thread to become ready to yield on the multiprocessor CPU. In rejecting these claims, the Examiner asserts that *Spix* discloses monitoring at col. 4, ll. 14-21. The cited text, however, concerns monitoring only general multithread execution with single-threaded CPU's. Since *Spix* at least fails to disclose the specific, claimed yield processes in connection with the claimed multithreaded CPU, Applicants respectfully submit that the Examiner has failed to establish anticipation of claims 6 and 21 by *Spix*. Reversal of the Examiner's rejections, and allowance of the claims, are therefore respectfully requested.

Dependent Claims 7 and 22

Dependent claims 7 and 22 respectively depend from claims 1 and 16, and each recites abandoning the yield on the multiprocessor CPU in response to detecting an event. As above, the Examiner relies on *Spix* for the rejection, and as above, the cited text at best concerns general yield processes in connection with a single-threaded CPU. Since *Spix* at least fails to disclose abandoning a yield call in the context of the claimed multithreaded CPU, Applicants respectfully submit that the Examiner has failed to establish anticipation of claims 7 and 22 by *Spix*. Reversal of the Examiner's rejections, and allowance of the claims, are therefore respectfully requested.

Dependent Claims 8 and 23

Dependent claims 8 and 23 respectively depend from claims 7 and 22, and each recites abandoning the yield on the multiprocessor CPU in response to detecting an a time-out or an external interrupt. Since *Spix* at least fails to disclose abandoning a yield call in the context of the claimed multithreaded CPU, let alone in response to detecting either a time-out or an interrupt, Applicants respectfully submit that the Examiner has failed to establish anticipation of claims 8 and 23 by *Spix*. Reversal of the Examiner's rejections, and allowance of the claims, are therefore respectfully requested.

Dependent Claims 9 and 24

Dependent claims 9 and 24 respectively depend from claims 7 and 22, and each recites returning control of the first thread in response to detecting the event. As above, the Examiner relies on *Spix* for the rejection, and as above, the cited text at best concerns general yield processes in connection with a single-threaded CPU. Since *Spix* at least fails to disclose returning control of the first thread in the context of the claimed multithreaded CPU, Applicants respectfully submit that the Examiner has failed to establish anticipation of claims 9 and 24 by *Spix*. Reversal of the Examiner's rejections, and allowance of the claims, are therefore respectfully requested.

Dependent Claims 10 and 25

Dependent claims 10 and 25 respectively depend from claims 9 and 24, and each recites saving the state of the operating system in response to detecting that at least the second thread is ready to yield on the multiprocessor CPU. As above, the Examiner relies on *Spix* for the rejection, and as above, the cited text at best concerns general yield processes in connection with a single-threaded CPU. Since *Spix* at least fails to saving a state in the context of the claimed multithreaded CPU, Applicants respectfully submit that the Examiner has failed to establish anticipation of claims 10 and 25 by *Spix*. Reversal of the Examiner's rejections, and allowance of the claims, are therefore respectfully requested.

Dependent Claims 11, 12, 26 and 27

Dependent claims 11, 12, 26 and 27 each regards idling a thread within a common virtual space in response to at least a second thread being ready to yield on the multithreaded CPU. In rejecting these claims, the Examiner asserts that *Spix* discloses idling threads at col. 8, ll. 61-67. The cited text, however, concerns idling threads in the context of single-threaded CPU's. Since *Spix* at least fails to disclose the specific, claimed yield processes in connection with the claimed multithreaded CPU, Applicants respectfully submit that the Examiner has failed to establish anticipation of claims 11, 12, 26 and 27 by *Spix*. Reversal of the Examiner's rejections, and allowance of the claims, are therefore respectfully requested.

Independent Claim 13

Independent claim 13 recites deferring a yield of a thread within a mulithreaded CPU system while at least a subset of the plurality of threads yield, and abandoning the yield of the thread in response to detecting an event while the yield is deferred. In rejecting this claim, the Examiner asserts that *Spix* discloses such processes at col. 38, ll. 41-48. As discussed above in connection with the rejection of claim 1, however, *Spix* fails to disclose or suggest the claimed yield call, let alone deferring a yield. Moreover, *Spix* does not disclose a multithreaded CPU. As such, Applicants respectfully request that the §102(e) rejection of claim 13 be reversed.

Moreover, Applicants respectfully submit that claim 13 is also non-obvious over *Spix* as there is no suggestion in the reference, or elsewhere in the cited prior art, of using a multithreaded CPU. The Examiner has provided no evidence of a motivation in the art to modify *Spix* to support deferring or abandoning yield commands for threads executing on a multithreaded CPU. The absence of such a motivation and suggestion is partially attributable to the peculiarities associated with multithreaded CPU's, which are not accommodated in the *Spix* system. As such, *Spix* would not motivate one of ordinary skill in the art to defer or abandon a yield of one thread on a multithreaded CPU based upon an event. Accordingly, claim 13 is also non-obvious over *Spix*. Reversal of the Examiner's rejections of claim 13 is therefore respectfully requested.

Dependent Claim 14

Dependent claim 14 depends from claim 13 and recites yielding the thread after the subset of threads yield, if the subset of threads yield prior to the event. The Examiner asserts that *Spix* discloses such features at col. 38, ll. 41-48. Again, however, the cited text at best concerns general yield processes in connection with a single-threaded CPU. Since *Spix* at least fails to disclose yielding a thread in the context of the claimed multithreaded CPU, Applicants respectfully submit that the Examiner has failed to establish anticipation of claim 14 by *Spix*.

Reversal of the Examiner's rejection, and allowance of the claim, are therefore respectfully requested.

B. Claim 15 is non-obvious over *Spix* and *Nakaya*.

Dependent claim 15 depends from claim 13 and recites that the event detected while the yield is deferred is selected from among a group consisting of: a time-out, an I/O interrupt and a combination thereof. Applicants' respectfully submit that claim 15 is non-obvious over *Spix* and *Nakaya* at least because there is no suggestion in the references, or elsewhere in the cited prior art, of using a multithreaded CPU. A *prima facie* showing of obviousness requires that the Examiner establish that the differences between a claimed invention and the prior art "are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art." 35 U.S.C. §103(a). Such a showing requires that all claimed features be disclosed or suggested by the prior art, along with objective evidence of the suggestion, teaching or motivation to combine or modify prior art references, as "[c]ombining prior art references without evidence of such a suggestion, teaching or motivation simply takes the inventor's disclosure as a blueprint for piecing together the prior art to defeat patentability -- the essence of hindsight." In re Dembiczak, 50 USPQ2d 1614, 1617 (Fed. Cir. 1999).

In the case of claim 15, the Examiner has provided no evidence of a motivation in the art to modify *Spix* to support yield commands for multithreaded CPU's, let alone to abandon a yield within multithreaded CPU in response to detecting a time-out or an I/O interrupt. The absence of such a motivation and suggestion is partially attributable to the peculiarities associated with multithreaded CPU's, which are not accommodated in the *Spix* system. *Spix* has no appreciation for the need to check the state of one thread in connection with allocating another thread to handle a particular task. As such, *Spix* would not motivate one of ordinary skill in the art to defer a yield of one thread on a multithreaded CPU based upon the state of another thread. *Nakaya* also fails to disclose or suggest a multithreaded CPU. As with *Spix*, *Nakaya* regards parallel processing with "serial" processors (Abstract). Accordingly, claim 1 is also non-obvious over *Spix* and *Nakaya*. Reversal of the Examiner's rejection of claim 15 is therefore respectfully requested.

VIII. CONCLUSION

In conclusion, Applicants respectfully request that the Board reverse the Examiner's rejections of claims 1-27 and 31-32, allow claim 33, and that the Application be passed to issue. If there are any questions regarding the foregoing, please contact the undersigned at 513/241-2324. Moreover, if any other charges or credits are necessary to complete this communication, please apply them to Deposit Account 23-3000.

Respectfully submitted,

WOOD, HERRON & EVANS, L.L.P.

Date: May 15, 2006

By: /Douglas A. Scholer/
Douglas A. Scholer
Reg. No. 52,197

2700 Carew Tower
441 Vine Street
Cincinnati, Ohio 45202
(513) 241-2324 - Telephone
(513) 241-6234 - Facsimile

IX. CLAIMS APPENDIX: CLAIMS ON APPEAL (S/N 09/939,235)

1. (Once Amended) A method for sharing resources on a multithreaded CPU capable of executing a plurality of threads, the method comprising:

deferring a yield of a first thread executing on the multithreaded CPU while waiting for at least a second thread executing on the multithreaded CPU to become ready to yield; and

yielding the first thread in response to at least the second thread becoming ready to yield.

2. (Original) The method according to claim 1, further comprising monitoring the plurality of threads for an occurrence.

3. (Original) The method according to claim 2, wherein the occurrence is a spin lock or an idle loop.

4. (Original) The method according to claim 2, further comprising making a yield call in response to the occurrence.

5. (Original) The method according to claim 1, further comprising marking storage of the first thread in response to receiving the yield call to indicate that the first thread is ready to yield.

6. (Original) The method according to claim 1, further comprising spinning the first thread while waiting for at least the second thread to become ready to yield.

7. (Original) The method according to claim 1, further comprising abandoning the yield call in response to detecting an event.

8. (Original) The method according to claim 7, wherein the event is a time-out or an external interrupt.

9. (Original) The method according to claim 7, further comprising returning control of the first thread to an operating system in response to detecting the event.

10. (Original) The method according to claim 9, further comprising saving the state of the operating system in response to detecting that at least the second thread is ready to yield.

11. (Original) The method according to claim 1, further comprising idling at least the first and second threads within a common virtual space in response to at least the second thread being ready to yield.

12. (Original) The method according to claim 11, further comprising idling all threads executing on the multithreaded CPU within the common virtual space.

13. (Once Amended) A method for yielding a thread within a multithreaded CPU data processing system, wherein each of a plurality of threads executing on a multithreaded CPU must execute within a common virtual space, the method comprising:

deferring a yield of a thread while at least a subset of the plurality of threads yield; and

abandoning the yield of the thread in response to detecting an event while the yield is deferred.

14. (Original) The method according to claim 13, further comprising yielding the thread after the subset of threads yield, if the subset of threads yield prior to the event.

15. (Original) The method according to claim 13, wherein the event is selected from among a group consisting of: a time-out, an I/O interrupt and a combination thereof.

16. (Original) An apparatus comprising:

a computer having a multithreaded CPU, wherein the CPU is configured to execute a plurality of threads; and

a program resident in the computer, the program configured to defer a yield of a first thread of the plurality while waiting for at least a second thread of the plurality to become ready to yield; and further to initiate the yield of the first thread in response to at least the second thread of the plurality becoming ready to yield.

17. (Original) The apparatus according to claim 16, wherein the program initiates monitoring the plurality of threads for an occurrence.

18. (Original) The method according to claim 17, wherein the occurrence is a spin lock or an idle loop.

19. (Original) The apparatus according to claim 17, wherein the program initiates a yield call in response to the occurrence.

20. (Original) The apparatus according to claim 16, wherein the program initiates marking storage of the first thread in response to receiving the yield call to indicate that the first thread is ready to yield.

21. (Original) The apparatus according to claim 16, wherein the program initiates spinning the first thread while waiting for at least the second thread of the plurality to become ready to yield.

22. (Original) The apparatus according to claim 16, wherein the program initiates abandoning the yield call in response to detecting an event.

23. (Original) The apparatus according to claim 22, wherein the event is a time-out or an external interrupt.

24. (Original) The apparatus according to claim 22, wherein the program initiates returning control of the first thread to an operating system in response to detecting the event.

25. (Original) The apparatus according to claim 24, wherein the program initiates saving the state of the operating system in response to detecting that at least the second thread is ready to yield.

26. (Original) The apparatus according to claim 16, wherein the program initiates idling at least the first and second threads of the plurality within a common virtual space in response to at least the second thread of the plurality being ready to yield.

27. (Original) The apparatus according to claim 26, wherein the program initiates idling all threads of the plurality of threads within the common virtual space.

28. - 30. (Cancelled)

31. (Original) A program product, comprising:

(a) a program for yielding a thread within a multithreaded CPU data processing system, wherein each of a plurality of threads that execute on a multithreaded CPU must execute within a common virtual space, wherein the program is configured to defer a yield of a first thread of the plurality while waiting for at least a second thread of the plurality to become ready to yield; and further to initiate the yield of the first thread in response to at least the second thread becoming ready to yield; and

(b) a signal bearing medium bearing the first program.

32. (Original) The program product of claim 31, wherein the signal bearing medium includes at least one of a recordable medium and a transmission-type medium.

33. (Added) The apparatus according to claim 16, wherein the program is configured to ensure that the plurality of threads execute within a common virtual space.

IX. EVIDENCE APPENDIX

09/939,235

None.

X. RELATED PROCEEDINGS APPENDIX

09/939,235

None.